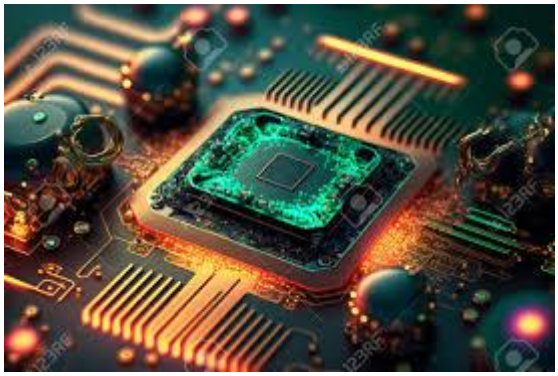
CENTRO UNIVERSITÁRIO DE VOTUPORANGA
ENGENHARIA DA COMPUTAÇÃO

ANDRÉ GUSTAVO HESPANHOL DE PAULA
RA:104416

TRABALHO PROCESSAMENTO DE IMAGENS

VOTUPORANGA – SP
2024

Imagens usadas:

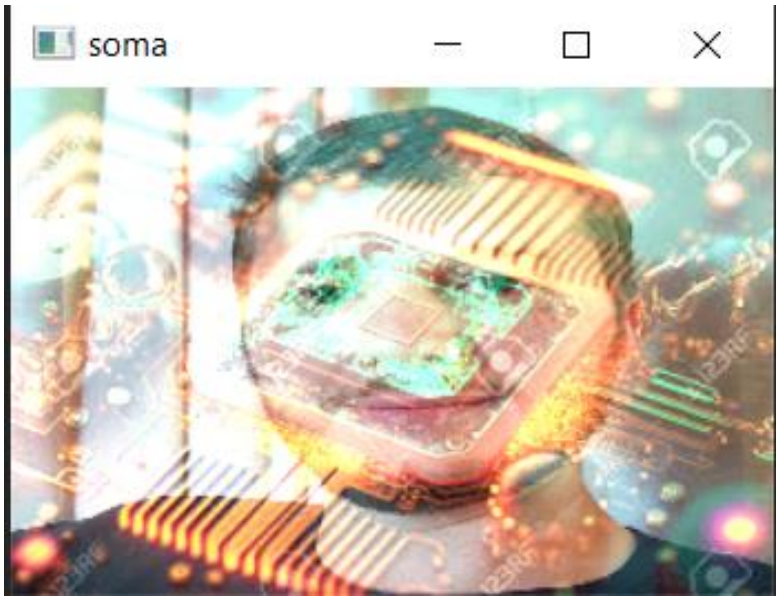


Adição:

Trabalho.py

```
1   import numpy as np
2   import cv2
3   img = cv2.imread('C:/Trabalho/eu.jpg')
4   img2 = cv2.imread('C:/Trabalho/placa.jpg')
5   height, width = img2.shape[:2]
6   img = cv2.resize(img,  dsize: (width, height))
7   cv2.imshow( winname: "imagem original", img)
8   cv2.imshow( winname: 'placa', img2)
9   soma = cv2.add(img, img2)
10  cv2.imshow( winname: 'soma', soma)
11  cv2.waitKey(0)
12  cv2.destroyAllWindows()
13
```

Resultado da adição:

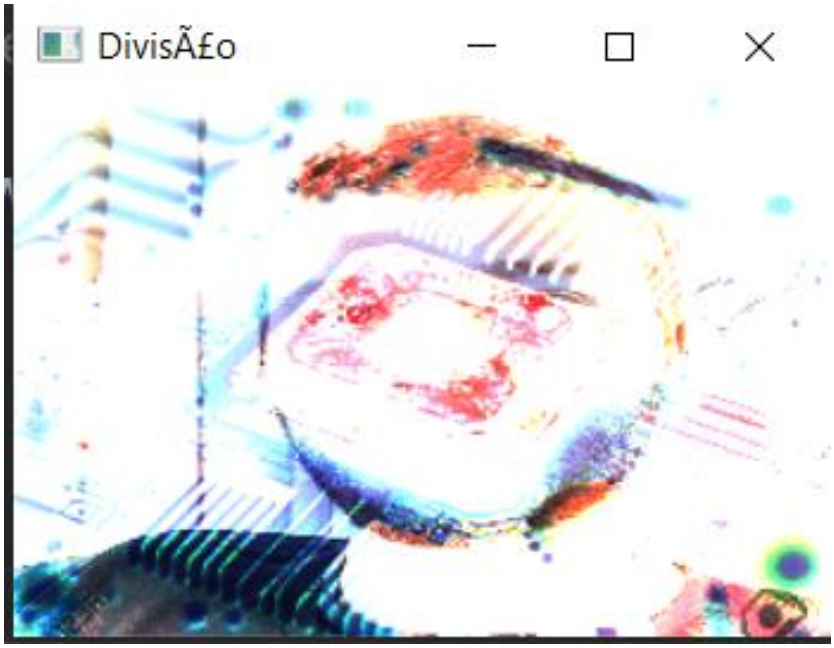


Nota se que a imagem da placa se sobrepõe sobre minha foto, em questão de qualidade é nítido que está bem poluída.

Divisão:

```
import cv2
import numpy as np
img = cv2.imread('C:/Trabalho/eu.jpg')
img2 = cv2.imread('C:/Trabalho/placa.jpg')
height, width = img2.shape[:2]
img = cv2.resize(img, dsize: (width, height))
img = img.astype(np.float32)
img2 = img2.astype(np.float32)
divisao = cv2.divide(img, img2 + np.finfo(float).eps)
divisao = np.clip(divisao * 255, a_min: 0, a_max: 255).astype(np.uint8)
cv2.imshow( winname: "Divisão", divisao)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Resultado da divisão:



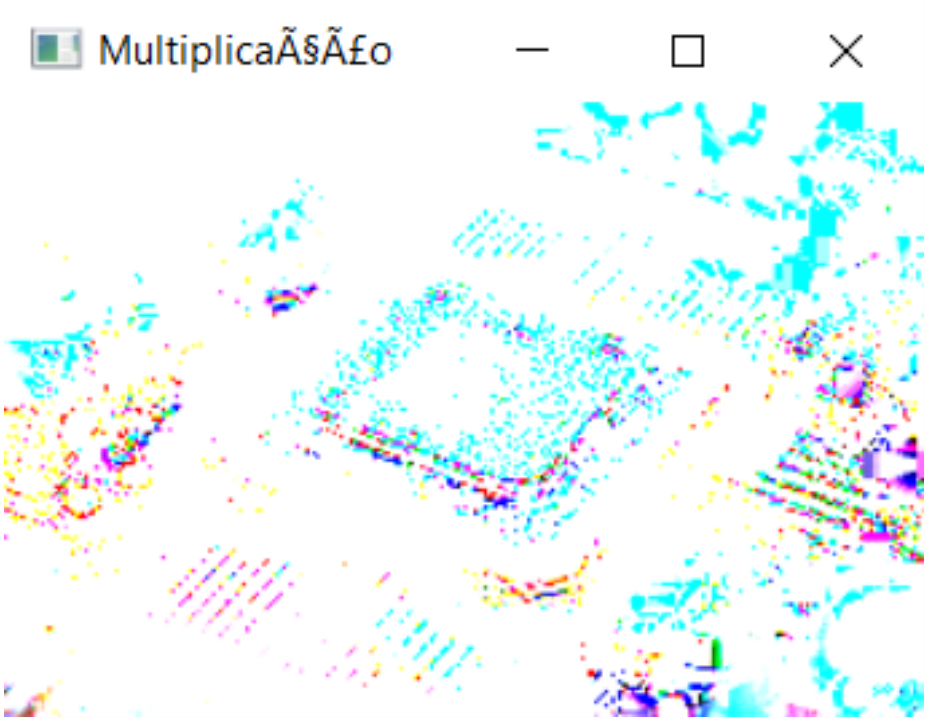
Com a operação de divisão a imagem fica quase irreconhecível, totalmente borrada e esbranquiçada.

Multiplicação:

Trabalho.py Teste.py

```
1  import cv2
2  img = cv2.imread('C:/Trabalho/eu.jpg')
3  img2 = cv2.imread('C:/Trabalho/placa.jpg')
4  height, width = img2.shape[:2]
5  img = cv2.resize(img, dsize: (width, height))
6  multiplicacao = cv2.multiply(img, img2)
7  cv2.imshow( winname: "Multiplicação", multiplicacao)
8  cv2.waitKey(0)
9  cv2.destroyAllWindows()
10
```

Resultado da multiplicação:



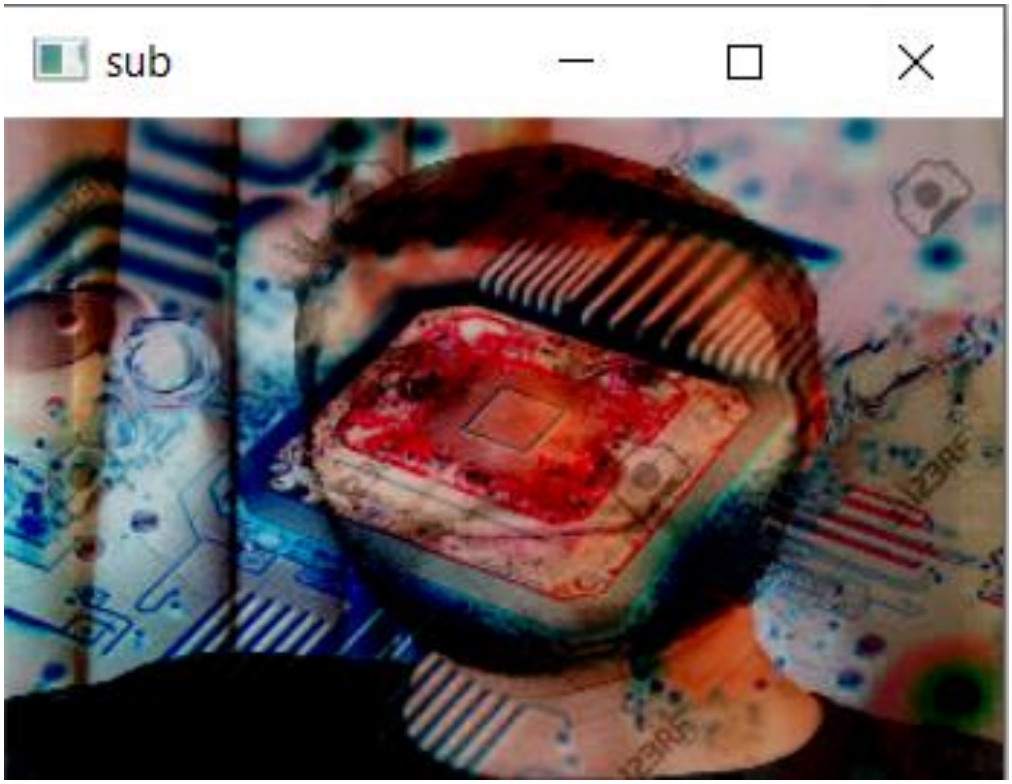
A imagem quase não é visível, pois a qualidade da imagem resultante da multiplicação depende das características das imagens de entrada e do contexto da aplicação.

Subtração:

Trabalho.py | Teste.py

```
1  import numpy as np
2  import cv2
3  img = cv2.imread('C:/Trabalho/eu.jpg')
4  img2 = cv2.imread('C:/Trabalho/placa.jpg')
5  height, width = img2.shape[:2]
6  img = cv2.resize(img, dsize: (width, height))
7  cv2.imshow( winname: "imagem original", img)
8  cv2.imshow( winname: 'placa', img2)
9  sub = cv2.subtract(img, img2)
10 cv2.imshow( winname: 'sub', sub)
11 cv2.waitKey(0)
12 cv2.destroyAllWindows()
13
```

Resultado da subtração:



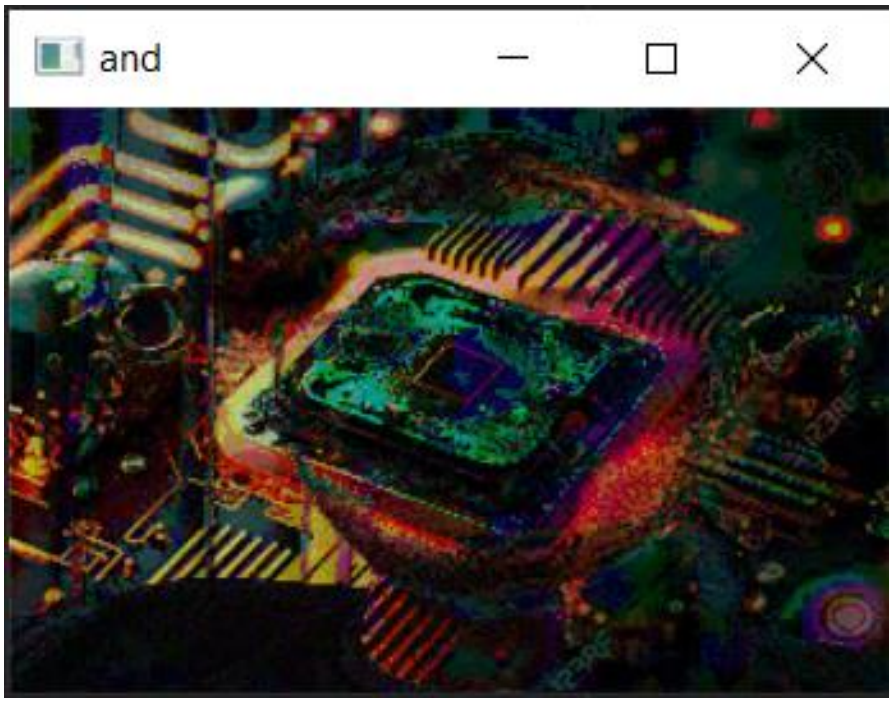
O resultado é a minha imagem com mais nitidez se comparada com a imagem da placa, o inverso da adição onde a imagem da placa se sobressai.

Operador AND:

Trabalho.py | Teste.py

```
1   import numpy as np
2   import cv2
3   img = cv2.imread('C:/Trabalho/eu.jpg')
4   img2 = cv2.imread('C:/Trabalho/placa.jpg')
5   height, width = img2.shape[:2]
6   img = cv2.resize(img, dsize: (width, height))
7   cv2.imshow( winname: "imagem original", img)
8   cv2.imshow( winname: 'placa', img2)
9   AND = cv2.bitwise_and(img, img2)
10  cv2.imshow( winname: 'and', AND)
11  cv2.waitKey(0)
12  cv2.destroyAllWindows()
13
```

Resultado:



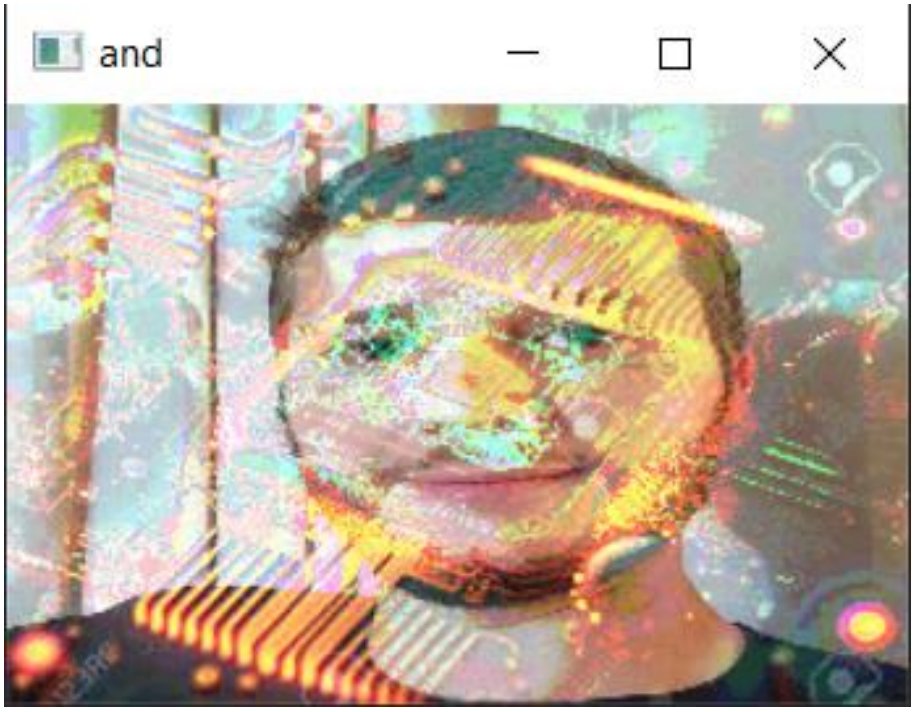
A operação AND é útil para extrair regiões de interesse comuns entre duas imagens, onde ambas as imagens têm pixels brancos nas mesmas posições.

Operador OR:

Trabalho.py | Teste.py

```
1  import numpy as np
2  import cv2
3  img = cv2.imread('C:/Trabalho/eu.jpg')
4  img2 = cv2.imread('C:/Trabalho/placa.jpg')
5  height, width = img2.shape[:2]
6  img = cv2.resize(img, dsize: (width, height))
7  cv2.imshow( winname: "imagem original", img)
8  cv2.imshow( winname: 'placa', img2)
9  OR = cv2.bitwise_or(img, img2)
10 cv2.imshow( winname: 'and', OR)
11 cv2.waitKey(0)
12 cv2.destroyAllWindows()
13
```

Resultado:



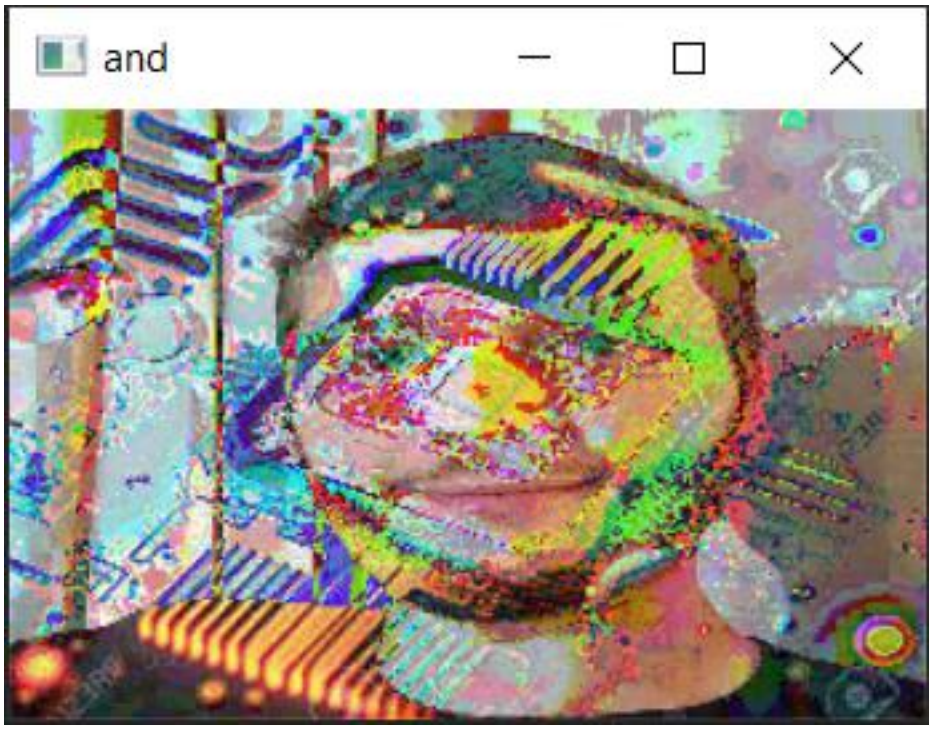
A operação OR entre duas imagens binárias resulta em uma nova imagem onde um pixel é branco (1) se pelo menos um dos pixels correspondentes nas duas imagens originais for branco (1).

Operador XOR:

Trabalho.py | Teste.py

```
1   import numpy as np
2   import cv2
3   img = cv2.imread('C:/Trabalho/eu.jpg')
4   img2 = cv2.imread('C:/Trabalho/placa.jpg')
5   height, width = img2.shape[:2]
6   img = cv2.resize(img, dsize: (width, height))
7   cv2.imshow( winname: "imagem original", img)
8   cv2.imshow( winname: 'placa', img2)
9   XOR = cv2.bitwise_xor(img, img2)
10  cv2.imshow( winname: 'and', XOR)
11  cv2.waitKey(0)
12  cv2.destroyAllWindows()
13
```

Resultado:



A operação XOR é útil para detectar áreas de diferença ou mudança entre duas imagens, destacando bordas e detalhes.

Operador NOT:

```
1   import numpy as np
2   import cv2
3   img = cv2.imread('C:/Trabalho/eu.jpg')
4   img2 = cv2.imread('C:/Trabalho/placa.jpg')
5   height, width = img2.shape[:2]
6   img = cv2.resize(img, dsize: (width, height))
7   cv2.imshow( winname: "imagem original", img)
8   cv2.imshow( winname: 'placa', img2)
9   NOT = cv2.bitwise_not(img, img2)
10  cv2.imshow( winname: 'not', NOT)
11  cv2.waitKey(0)
12  cv2.destroyAllWindows()
13
```

Resultado:

A imagem ficou em negativo, retirando qualquer rastro da imagem da placa. Se mudar a ordem no código a imagem em destaque será da placa.

Segmento e mascara:

Trabalho.py | Teste.py

```
1   import numpy as np
2   import cv2
3   img = cv2.imread('C:/Trabalho/placa.jpg')
4   cv2.imshow( winname: "imagem original", img)
5   recorte = img[50:200, 50:200]
6   cv2.imshow( winname: 'recorte', recorte)
7   m1 = np.zeros(img.shape[:2], dtype="uint8")
8   (cx, cy) = (img.shape[1]//2, img.shape[1]//2)
9   cv2.circle(m1, center: (cx, cy), radius: 90, color: 100, -1)
10  m2 = cv2.bitwise_and(img, img, mask=m1)
11  cv2.imshow( winname: 'Mascara', m2)
12  cv2.waitKey(0)
13  cv2.destroyAllWindows()
14
```